PREVIO PRÁCTICA 3 – PSCD

Grupo Miércoles A-G1: 12:00-14:00
-Práctica 3-

Autores y NIA:
Ignacio Ortega Lalmolda 610720
Héctor Toral Pallás 798095

Ejercicio 1:

1. Descripción datos compartidos, enumeración procesos, sincronización:

Procesos usuarios y proceso conductor:

El conductor abre puertas y ESPERA a que el tren esté lleno (mirando valores contadores ocupación), llenándose primero el vagón nuevo w1 y luego el antiguo w2.

En cuanto los vagones w1 y w2 se llenan, el conductor **cierra las puertas de entrada** y sale de viaje.

Una vez que llegamos del viaje, el conductor **abre puertas las puertas de salida** y los usuarios van saliendo. Para saber de qué vagón salimos hemos pensado en hacer un booleano local llamado wag1 a cada proceso que nos indicará en que vagón hemos entrado siendo true si subimos a w1 y false si subimos a w2.

Una vez que han salido todos, el conductor **cierra las puertas de salida** y **abre las de entrada** y repetimos viaje hasta completar el número máximo de viajes.

1- Recursos compartidos:

Las variables de tipo booleano que indica si las puertas del vagón están abiertas o cerradas para que puedan entrar o salir los pasajeros del vagón	*boolean ptEntradaAbiertas* *boolean ptSalidaAbiertas*
La capacidad del vagón, aunque es constante y no va cambiar la emplean ambos procesos.	*N_W1 N_W2*
Las variables de tipo entero que indican si los vagones llevan pasajeros o están vacíos	*w1, w2*

2- Implementación con await en pseudocódigo:

```
const integer N := 48, const integer N_VIAJES := 8
const integer N_W1 := 4, const integer N_W2 := 2
integer w1 := 0, integer w2 := 0
integer hanBajado := 0,
boolean ptEntradaAbiertas := true, boolean ptSalidaAbiertas := false,
```

```
Process usuario [i], (i: 1..N)::
    bool wag1
    <await (ptEntradaAbiertas = true ∧ (w1 <
    N_W1 v w2 < N_W2))
        // Montar primero en el vagón w1
        if (w1 < N_W1)
            // Saber donde monto
            wag1 := true
            w1++;
        end
        // Si w1 estaba lleno nos vamos en w2
        else
            wag1 := false
            numWag[i] = 2
            w2++;
        end
        // ADD_EVENT("MONTA",...

    >

    // Ya estaba montado
    <await (ptSalidaAbiertas = true)
        // ¿De qué vagón me bajo?
        if (wag1 = true)
            w1--
            hanBajado +:= 1
        end
        else
            w2--
            hanBajado +:= 1
        end
         //ADD_EVENT...

    >
```

```
Process conductor::
    for i:=1..N_VIAJES
        <await (w1 = N_W1 ∧ w2 = N_W2)
            ptEntradaAbiertas := false
            // ADD_EVENT...
        >

        //No es atómico pero no importa porque
        los ususarios están bloqueados

        // viajar (tiempo aleatorio)
        sleep(viaje)
        ptSalidaAbiertas := true
        //ADD_EVENT...

        // esperar a que todos bajen
        <await (w1 = 0 ∧ w2 =0)
            ptSalidaAbiertas := false
            ptEntradaAbiertas := true
            // ADD_EVENT...
        >
        i++
    end
end
```

2. Esbozo del código:

1- Implementación del await con el paso de testigo entre pseudocódigo y c++:

```
const integer N := 48, const integer N_VIAJES := 8
const integer N_W1 := 4, const integer N_W2 := 2
integer w1 := 0, integer w2 := 0, integer hanBajado := 0,
boolean ptEntradaAbiertas := true, boolean ptSalidaAbiertas := false,
integer array[1,N] numWag := (1..N, 0)

// Para las guardas de los usuarios, b's asociados a cada guarda, d's para cada b
semaphore array [1,N]b_1 := (1..N,0), integer array [1,N]d_1 := (1..N,0)
semaphore array [1,N]b_2 := (1..N,0), integer array [1,N]d_2 := (1..N,0)

// Para las guardas del conductor, b's asociados a cada guarda, d's para cada b
semaphore array [1, N_ VIAJES]b_3 := (1.. N_ VIAJES,0), integer array [1,N_ VIAJES]d_3 := (1..N_VIAJES,0)
semaphore array [1, N_ VIAJES]b_4 := (1..N_ VIAJES,0), integer array [1,N_ VIAJES]d_4 := (1..N_VIAJES,0)
semaphore mutex := 1
```

```
Process usuario [i] {
    bool wag1;
    wait(mutex);
        if !(ptEntradaAbiertas && (w1 < N_W1
        || w2 < N_W2)) {
            d_1[i]++;
            signal(mutex);
            wait(b_1[i]);
        }
        // Montar primero en el vagón w1
        if (w1 < N_W1) {
           // Saber donde monto
           wag1 = true;
           w1++;
           // ADD_EVENT("MONTA",...
        } else {
        // Si w1 estaba lleno nos vamos en w2
           wag1 = false;
           w2++;
           // ADD_EVENT("MONTA",...
        }
        AVISAR_1(i);
    // Ya estaba montado
    wait(mutex);
        if !(ptSalidaAbiertas) {
            d[i]2++;
            signal(mutex);
            wait(b[i]2);
        }
        // ¿De qué vagón me bajo?
        if (wag1) {
            w1--;
            hanBajado ++;
            //ADD_EVENT
        } else {
            w2--;
            hanBajado ++;
            //ADD_EVENT
        }
        AVISAR_2(i);
    }
```

```
Process conductor {
    for (int i =1; i <= N_VIAJES; i++) {
        wait(mutex);
            if !(w1 == N_W1 &&  w2 == N_W2)) {
                d_3[i]++;
                signal(mutex);
                wait(b_3[i]);
            }
            ptEntradaAbiertas = false;
            //ADD_EVENT...
            AVISAR_3(i);

            sleep(viaje);
            ptSalidaAbiertas = true;
            //ADD_EVENT...

        wait(mutex);
            if !(w1 == 0 && w2 ==0) {
                d4[i]++;
                signal(mutex);
                wait(b4[i]);
            }
            ptSalidaAbiertas = false;
            ptEntradaAbiertas = true;
            //ADD_EVENT...
            AVISAR_4(i);
    }
}
```

2- Implementación del avisar en pseudocódigo

operation AVISAR_1(integer i)::
//Hay n casos, y no podemos hacer el mismo switch, soy el filosofo iesimo

```
int j = i+1%N                                   // Empezar en el sig para no dar prioridad al user i
                                                // Si no empezar por el primero j = 1%N
bool hayHueco = false
while not (hayHueco ∧ j < i)                    //El actual no miro. Para parar o si primero j <= N
      hayHueco := (ptEntradaAbiertas ∧ (w1 < N_W1 ∨ w2 < N_W2)
   if hayHueco
   d_1[j]--
   signal(b_1[j])
   else
   j := j+1%N
   end if
end while
if not hayHueco
   signal(mutex)
end if
```

operation AVISAR_2(integer i)::
//Hay n casos, y no podemos hacer el mismo switch, soy el filosofo iesimo

```
int j = i+1%N                                   // Empezar en el sig para no dar prioridad al user i
                                                // Si no empezar por el primero j = 1%N
bool abiertas = false
while not abiertas ∧ j < i                      //El actual no miro. Para parar o si primero j <= N
      abiertas := (ptSalidaAbiertas)
   if abiertas
   d_2[j]--
   signal(b_2[j])
   else
   j := j+1%N
   end if
end while
if not abiertas
   signal(mutex)
end if
```

operation AVISAR_3(integer i)::
//Hay n casos, y no podemos hacer el mismo switch, soy el filosofo iesimo

```
int j = i+1%N_VIAJES                    // Empezar en el sig para no dar prioridad al user i
                                        // Si no empezar por el primero j = 1%N_VIAJES
bool lleno = false
while not lleno ∧ j < i                 //El actual no miro. Para parar o si primero j <= N_VIAJES
      lleno := (w1 = N_W1 ∧ w2 = N_W2)
   if lleno
   d_3[j]--
   signal(b_3[j])
   else
   j := j+1%N
   end if
end while
if not lleno
   signal(mutex)
end if
```

operation AVISAR_4(integer i)::
//Hay n casos, y no podemos hacer el mismo switch, soy el filosofo iesimo

```
int j = i+1%N_VIAJES                    // Empezar en el sig para no dar prioridad al user i
                                        // Si no empezar por el primero j = 1%N
bool estaVacio = false
while not estaVacio ∧ j < i             //El actual no miro. Para parar o si primero j <= N
      estaVacio := (w1 = 0 ∧ w2 =0)
   if hayHueco
   d_4[j]--
   signal(b_4[j])
   else
   j := j+1%N
   end if
end while
if not estaVacio
   signal(mutex)
end if
```